Sheet 5 solution

1. This deformations occurs because the aspect ratio of the clipping window on the projection plan does not have the same aspect ratio as the display window or the current viewport
   To correct this problem we can do on of the following
   - Using the default viewport (the entire window) and making the aspect ratio of the window the same as the aspect ratio of the clipping window by changing the window size or changing the size of the clipping window
   - In case of drawing on a portion of the screen window using a viewport, we must choose the size of the viewport and the size of the clipping window so that they have the same aspect ratio.

   The function used to set a view port is as follows:

   ```
   void glViewport(GLint x, GLint y, GLsizei w, GLsizei h)
   ```

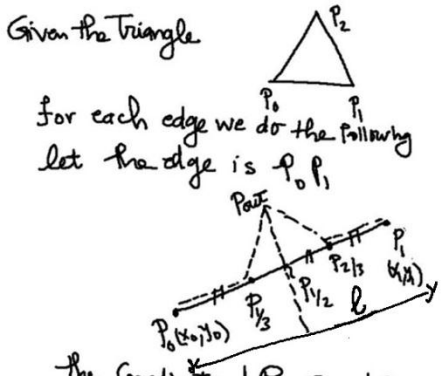
   The function used to set the size of the window is as follows

   ```
   glutInitWindowSize(640, 480);
   ```

2.

```
// ViewPortIn2D.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>

const int N=500,M=500;
void drawDampedSine(void)
{
    GLdouble x,func;
    glBegin(GL_LINES);
    glVertex2d(-2,0);
    glVertex2d(2,0);
    glVertex2d(0,-2);
    glVertex2d(0,2);
    glEnd();
    glBegin(GL_POINTS);
    for(x=0;x<4;x+=0.005)
    {
        func=exp(-fabs(x))*cos(2*3.14159265*x);
        glVertex2d(x,func);
    }
    glEnd();
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    glPointSize(2);
    // upper left quarter
    glViewport((int)(0.1*N), (int)(0.55*M), (int)(0.35*N), (int)(0.35*M));
    drawDampedSine();
    // down left quarter
    glViewport((int)(0.1*N), (int)(0.1*M), (int)(0.35*N), (int)(0.35*M));
    drawDampedSine();
    // upper right quarter
    glViewport((int)(0.55*N), (int)(0.55*M), (int)(0.35*N), (int)(0.35*M));
    drawDampedSine();
    // down right quarter
    glViewport((int)(0.55*N), (int)(0.1*M), (int)(0.35*N), (int)(0.35*M));
    drawDampedSine();
    glFlush();
}

void myinit()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-4.0, 4.0, -4.0, 4.0);
    glMatrixMode(GL_MODELVIEW);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glColor3f(0.0,0.0,0.0);
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(N, M);
    glutCreateWindow("View port and 2D graphics");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
}


```

3.



for each edge we do the following

let the edge is P_0P_1

the Coordinates of $P_{1/3}$, $P_{1/2}$ and $P_{2/3}$ Can be calculated as flows

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$$

$t = 1/3 \rightarrow P_{1/3}$

$t = 1/2 \rightarrow P_{1/2}$

$t = 2/3 \rightarrow P_{2/3}$

then we find the perpendiculer to the vector $\overline{x_0x_1}$

$$\overline{x_0x_1} = (x_1 - x_0)\overline{ax} + (y_1 - y_0)\overline{ay}$$

$$\perp \overline{x_0x} = -(y_1 - y_0)\overline{ax} + (x_1 - x_0)\overline{ay} \quad \checkmark$$

$$\text{or} = (y_1 - y_0)\overline{ax} - (x_1 - x_0)\overline{ay}$$

we use the first

the unit vector of the $\perp$ is

$$\frac{-(y_1 - y_0)\overline{ax} + (x_1 - x_0)\overline{ay}}{\sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2}}$$

to find the coordinates of P_{out} we add a displacement of $\frac{l}{2\sqrt{3}}$ to $P_{1/2}$ in the $\perp$ direction

$$x_{out} = x_{1/2} + \frac{-(y_1 - y_0)\frac{l}{2\sqrt{3}}}{\sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2}}$$

$$y_{out} = y_{1/2} + \frac{(x_1 - x_0)\frac{l}{2\sqrt{3}}}{\sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2}}$$

Now we have all the required point to draw the required line or to recurssively subdivide each line up to a specified number of sub. division

```
// Prob6Ch2Angle5ED.cpp : Defines the entry point for the console application.
#include "stdafx.h"
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>
// initial triangle
// number of recurrsive line divisions
const int N=1;
const int perpendicular=0;// 0 for the first (Up) 1 for the second (Down)
void processLine(GLfloat *P0, GLfloat *P1, int n)
{
    GLfloat firstX,firstY,thirdX,thirdY,middleX,middleY,twoThirdsX,twoThirdsY,outX,outY;

    // getting the coordinated of interested points on the line
    firstX=P0[0];
    firstY=P0[1];
    thirdX=P0[0]+(1.0F/3)*(P1[0]-P0[0]);
    thirdY=P0[1]+(1.0F/3)*(P1[1]-P0[1]);
    middleX=P0[0]+(1.0F/2)*(P1[0]-P0[0]);
    middleY=P0[1]+(1.0F/2)*(P1[1]-P0[1]);
    twoThirdsX=P0[0]+(2.0F/3)*(P1[0]-P0[0]);
    twoThirdsY=P0[1]+(2.0F/3)*(P1[1]-P0[1]);
    // the vector from P0 to P1 is: (p1X-p0X) ax + (P1Y-p0Y) ay
    // the perpendicular is       :-(p1Y-p0Y) ax + (p1X-p0X) ay (First)
    //                    or      : (p1Y-p0Y) ax - (p1X-p0X) ay (Second)

    // finding the unit vector of the vector perpendicular to the vector P0 to P1
    GLfloat unitPerpendicularX,unitPerpendicularY;
    GLfloat lineLength=sqrt((P1[1]-P0[1])*(P1[1]-P0[1])+((P1[0]-P0[0])*(P1[0]-P0[0])));
    if(perpendicular==0)
    {
        unitPerpendicularX=(-(P1[1]-P0[1]))/lineLength;
        unitPerpendicularY=((P1[0]-P0[0]))/lineLength;
    }
    else
    {
        unitPerpendicularX=((P1[1]-P0[1]))/lineLength;
        unitPerpendicularY=(-(P1[0]-P0[0]))/lineLength;
    }
    // find the coordinated of out of line point by adding a displacement to the middle
    point
    outX=middleX+unitPerpendicularX*(lineLength/(sqrt(2.0F)*3));
    outY=middleY+unitPerpendicularY*(lineLength/(sqrt(2.0F)*3));
    if(n>0)
    {
        // redo for the four resulting line segements
        GLfloat newLine1P0[2];newLine1P0[0]=P0[0];newLine1P0[1]=P0[1];
        GLfloat newLine1P1[2];newLine1P1[0]=thirdX;newLine1P1[1]=thirdY;
        processLine(newLine1P0,newLine1P1,n-1);

        GLfloat newLine2P0[2];newLine2P0[0]=thirdX;newLine2P0[1]=thirdY;
        GLfloat newLine2P1[2];newLine2P1[0]=outX;newLine2P1[1]=outY;
        processLine(newLine2P0,newLine2P1,n-1);

        GLfloat newLine3P0[2];newLine3P0[0]=outX;newLine3P0[1]=outY;
        GLfloat newLine3P1[2];newLine3P1[0]=twoThirdsX;newLine3P1[1]=twoThirdsY;
        processLine(newLine3P0,newLine3P1,n-1);

        GLfloat newLine4P0[2];newLine4P0[0]=twoThirdsX;newLine4P0[1]=twoThirdsY;
        GLfloat newLine4P1[2];newLine4P1[0]=P1[0];newLine4P1[1]=P1[1];
        processLine(newLine4P0,newLine4P1,n-1);
    }
    else
    {
        glVertex2fv(P0);
        glVertex2f(thirdX,thirdY);
        glVertex2f(thirdX,thirdY);
        glVertex2f(outX,outY);
        glVertex2f(outX,outY);
        glVertex2f(twoThirdsX,twoThirdsY);
```

```
70          glVertex2f(twoThirdsX,twoThirdsY);
71          glVertex2fv(P1);
72      }
73  }
74  void display()
75  {
76      glClear(GL_COLOR_BUFFER_BIT);
77      glBegin(GL_LINES);
78      // original equilateral triangle
79      GLfloat P0[2];P0[0]=0;P0[1]=0;
80      GLfloat P1[2];P1[0]=1.8F;P1[1]=0;
81      GLfloat P2[2];P2[0]=0.9F;P2[1]=1.8F*sqrt(3.0F)/2.0F;
82      processLine(P0,P1,N);
83      processLine(P1,P2,N);
84      processLine(P2,P0,N);
85      glEnd();
86      glFlush();
87  }
88
89  void myinit()
90  {
91      glMatrixMode(GL_PROJECTION);
92      glLoadIdentity();
93      gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
94      glMatrixMode(GL_MODELVIEW);
95      glClearColor (1.0, 1.0, 1.0, 1.0);
96      glColor3f(0.0,0.0,0.0);
97  }
98
99  int main(int argc, char **argv)
100 {
101     glutInit(&argc, argv);
102     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
103     glutInitWindowSize(500, 500);
104     glutCreateWindow("Space filling");
105     glutDisplayFunc(display);
106     myinit();
107     glutMainLoop();
108 }
```